iPART : An Automated Phase Analysis and Recognition Tool

Bob Davies Jean-Yves Bouguet Marzia Polito Murali M. Annavaram

Microprocessor Research Labs - Intel Corporation - SC12-303
Santa Clara, CA 95052, USA

{bob.davies,jean-yves.bouguet,marzia.polito,murali.m.annavaram}@intel.com

Abstract

Detecting and exploiting program's cyclic phase behavior can reduce program simulation time by eliminating simulation of repeated phases. To use this technique over a wide range of complex applications, this paper presents a novel non-intrusive, and low overhead tool, *iPART*, for analyzing and recognizing phase behavior of programs running on native hardware. iPART is a fully-automated, configurable tool built on top of Intel's VTune™ performance analyzer.

iPART samples program execution in real-time and collects Extended Instruction Pointers (EIP) and several program metrics via embedded performance counters. While iPART is capable of using three different clustering algorithms to detect program phases, in this paper phases are identified by clustering EIP vectors (each EIPV has 100 million instructions) using the k-means algorithm. Based on experimental data this paper proposes two major enhancements to previously proposed phase detection techniques using basic block vectors. First, this paper shows that large intra-phase CPI (Cycles Per Instruction) variance in some applications reduces the confidence in CPI estimation. It shows that *stratified sampling*, which optimally allocates one or more simulation points per phase, is an effective solution to improve the confidence. Second, this paper also shows that rigidly building vectors of 100 million instructions produces ambiguous vectors at phase boundaries that will have EIPs from adjoining phases. This paper present *QPhase* as a novel approach to split EIPVs at phase boundaries into smaller vectors. QPhase tracks the locus of execution to split code vectors whenever the locus changes dramatically.

Experimental results show that on average over the entire Spec2k benchmark suite, our two proposed iPART estimators (non stratified and stratified) achieve CPI estimation errors 3 times smaller than the conventional uniform sampling technique. The benefit of the novel stratified sampling estimator is most noticeable for applications exhibiting large intra-phase CPI variance. In those cases stratified iPART produces CPI estimates with increased confidence levels of by a typical factor of 2.

Over the entire SPEC2k benchmark, QPhase produces almost 9% fewer phase transitions while preserving the same accuracy. In addition, phase widths are increased by an average of almost 150 million instructions, making phases an easier target for trace activity.

1 Introduction

Processor designers rely heavily on representative benchmark simulations to evaluate various design alternatives. Hence, significant emphasis is placed on accurately modeling the design choices in software simulators. Accurate modeling of a complex design may dramatically reduce simulation speed, in spite of the increasing processing power, thereby restricting the ability to study design tradeoffs. Researchers use ad-hoc solutions, such as simulating only a small fraction of the overall benchmark execution in the hope that the simulated fraction is a good representation of the overall behavior. However, recent studies [14, 3, 4] have shown that programs exhibit different behaviors during different execution phases that occur over a long time period. Consequently, there is an opposing trend between the desire for accurate simulations and the need to simulate benchmarks over a long period of time to capture the phase behavior.

One way to reduce the simulation time without compromising accuracy is to simulate several samples of program execution. Recently, Wunderlich *et al.* [17] proposed simulating a large number of samples each with a small sample size to capture the overall behavior of a program. Using statistical analysis they computed the optimal number of samples and the sample size required to achieve a given error margin with high confidence. Sherwood *et al.* [14] proposed a phase-based sampling approach. They observed that programs exhibit cyclic behavior across many different performance metrics. Hence simulating only one sample from each phase with a large sample size can improve simulation speed without unduly reducing accuracy. However, their analysis is based on code instrumentation and simulations, which restricts our ability to apply the methodology to a wide-range of applications running on complex native hardware.

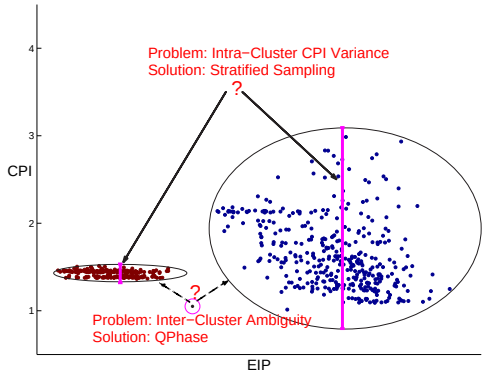

Fig. 1: **Simplified illustration of code clustering difficulties on fma3d.** The cluster on the right has high intra-cluster CPI variance, resulting in a low confidence in CPI estimation when simulating one-sample-per-phase. Stratified sampling addresses this difficulty. The isolated EIP in the center shows the inter-cluster ambiguity at phase boundaries.QPhase addresses the second difficulty.

In this paper we present a novel, non-intrusive, and low overhead methodology for detecting and analyzing phase behavior of programs running on native hardware. The result is *iPART (an Automated Phase Analysis and Recognition Tool)*, a fully-automated tool that is built on top of Intel's VTune™ performance analyzer.

iPART detects phases using a two step process: in the first step it samples program execution and collects Extended Instruction Pointers (EIP), and several performance metrics using processor performance counters. In the second step it identifies program phases by clustering EIP vectors (EIPVs) obtained from the sampled VTune data. iPART is capable of clustering EIPVs using three different clustering algorithms, namely, k-means [7, 8], Spectral [11] and Agglomerative [9, 8]. The three clustering algorithms perform equally well; hence, in this paper we present clustering results using only k-means, which closely follows the approach suggested by Sherwood [14]. Phase identification provides an accurate

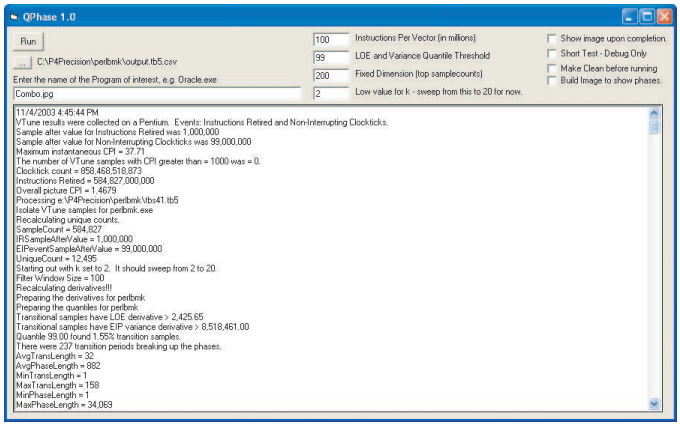

Fig. 2: **Screen shot of the iPART tool in action:** The tool is fully automatic and allows one to run the measurements and analysis directly.

estimation of Cycles Per Instruction (CPI) using only a small portion of code. The error and confidence of the CPI estimation is computed in a subsequent validation phase.

This paper makes three major contributions in the areas of phase analysis and recognition methodology.

1. iPART leverages the embedded hardware counters available on Intel™ processors that track processor activity. These sample counters are collected in real time during program execution without code instrumentation and with negligible overhead.

2. Our experimental results show that some applications exhibit large intra-phase CPI variance using k-means clustering. Figure 1 shows the two clusters identified by k-means for fma3d, a Spec2k benchmark. The cluster on the left hand has a small CPI variance and hence using one-sample-per-phase approach suggested in previous work [14] captures the entire phase behavior accurately. Unfortunately, the cluster on the right has a much larger CPI variance. While one may be able to pick a single sample with minimal CPI estimation error, the variance in the CPI estimation error for that phase will be high thereby reducing the confidence. To solve the intra-phase variance problem, this paper proposes *stratified sampling* as an effective alternative. Stratified sampling optimally allocates a fixed total budget of samples to each program phase according to the relative variability of CPI within each phase. Using *stratified sampling* the confidence in CPI estimation can be improved by more than 80% over the best known previous approaches.

3. Finally, this paper shows that the rigid approach of always building vectors with 100 million instructions will inevitably produce ambiguous vectors at phase boundaries that will have some EIPs from the previous phase and some EIPs from the next phase. For instance, in Figure 1 the isolated EIPV in the center fits neither in the first phase nor in the second phase. The clustering algorithm either treats this to be part of a different phase, thereby increasing the k value, or may place these vectors into one of the existing phases, thereby increasing the error in CPI estimation. This paper presents *QPhase*, a novel approach to split 100 million EIPVs at phase boundaries into smaller vectors by tracking the locus of execution to identify phase changes.

The rest of this paper is organized as follows: Section 2 compares iPART with the most relevant prior work done in the area of analyzing program behavior. Section 3 introduces the iPART tool, that captures raw data using VTune. Section 4 describes the overall methodology used to identify phases, and is followed by Section 5 that reports results of two different performance estimators. Section 6 presents QPhase, a technique that enhances phase detection and finally Section 7 concludes and presents directions for future research.

2 Previous Work

Researchers have proposed several techniques to detect and exploit phase behavior of programs [1, 12, 14, 3, 16, 17, 5]. In this section we focus only on the most relevant prior work and provide a qualitative comparison of our approach with previous approaches.

KlienOsowski and Lilja [10] proposed *MinneSPEC*, which provides a modified input set for the Spec2k benchmark suite. MinneSPEC tries to reduce the execution time of Spec2k benchmarks with smaller input data sets without significantly altering the dynamic instruction mix and the function call sequences. However, there are limitations to MinneSPEC. For instance, evaluating architectural techniques that are sensitive to memory behavior can produce substantially different results with MinneSPEC than with the *reference* input set of Spec2k benchmarks.

Sherwood *et al.* [14, 16] proposed phase-based sampling which simulates only a small number of (about 10) code segments of each 10-100 million instructions. They used the execution frequency of basic blocks to form basic block vectors (BBV) over consecutive segments of 100M instructions. The BBVs are then grouped in k clusters using k-means clustering technique. Each cluster represents a unique program phase which is repeated a number of times throughout the execution. The results from this analysis are then applied to improve efficiency by simulating only one sample from each phase. The advantage of their approach is that it is mostly machine independent; however, phase detection is a slow and tedious task requiring simulations and code instrumentation for identifying basic blocks and processing long instruction traces to build BBVs. In this paper, we propose to substitute BBVs with EIPVs which can be automatically obtained by the iPART tool using VTune sampling. Our results show that EIPVs and BBVs are equally suited to detect program phases.

Eeckhout *et al.* [6] used statistical modelling to efficiently explore the workload space in order to reduce simulation time. Their analysis is based on collecting several characteristics, such as instruction mix, and cache misses, over the entire program execution and then using principal component analysis to determine a reduced set of orthogonal parameters. Using this analysis they infer which program input pairs have similar behavior and prune redundant pairs for efficient design exploration.

Wunderlich *et al.* [17] also used statistical approaches to reduce simulation time. Their strategy consists of taking numerous samples (~ 50,000) of small size (1,000 instructions)

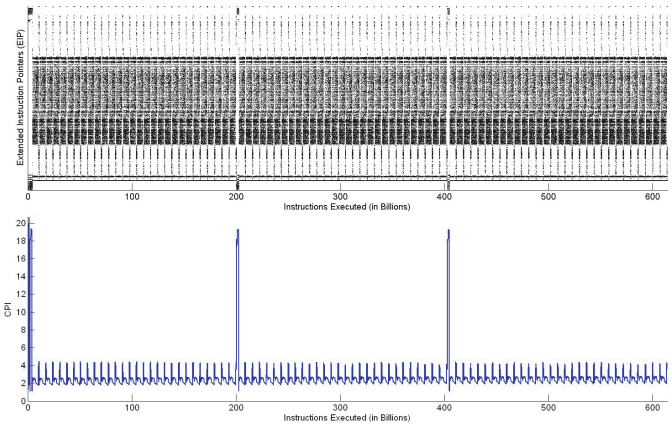

Fig. 3: **ammp:** This image represents the code sequence recorded by VTune during the execution of the Spec2k benchmark ammp on a Pentium 4 machine. For every one million instructions executed, VTune interrupts the code and records the Extended Instruction Pointer (EIP) of the instruction that is about to execute. Every dot in this image represents one such recorded EIP. Each row in the image corresponds to a unique EIP (there are $N = 3090$ in the image), and each column corresponds to a single sample acquired by VTune (therefore, there is only one dot per column.) During execution time, VTune also monitors clocktick count. This allows for a precise monitoring of the instantaneous CPI shown in the bottom figure. Observe that patterns of variations of CPI are strongly temporally correlated with the code sequence behavior in EIP space. This suggests EIP is a valid attribute for extracting program phases. This observation is at the heart of the method presented in the paper.

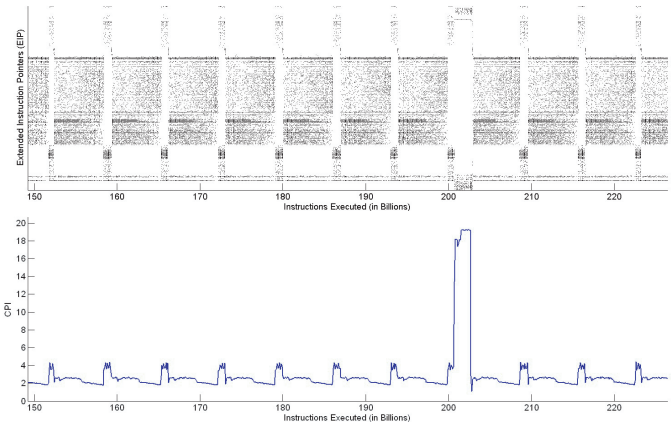

Fig. 4: **ammp:** A close up view of Figure 3 showing a segment of 80 billion executed instructions of ammp. Strong temporal correlations between EIP space and CPI space are also noticeable at that scale.

at regular intervals during program execution. The paper presents a systematic way to compute the optimal frequency of sampling and the size of each sample that guarantees a specified error bound for the Instructions Per Cycle (IPC) estimate. This approach uses IPC as the only attribute to determine sampling frequency, i.e. it is equivalent to a frequency analysis of the temporal IPC signal. Hence the accuracy of approximation is directly controlled by the temporal IPC curve. One attribute of this approach is the fact that the output of the reduction process consists of a very large number of code segments (several tens of thousands) compared to only a few samples proposed by Sherwood. Another issue is that since no machine independent attribute is used to drive the analysis, the sampling results obtained on one machine may not generalize to another machine.

Balasubramonian *et al.* [1] exploited program phase behavior to dynamically reconfigure the processor to meet application demands. Dhodapkar and Smith [3] also identified program phase changes and represented each phase by a working set signature. These working set signatures are stored in a history table to recognize repeating signatures, which are then removed from design space exploration.

Duesterwald *et al.* [4] proposed monitoring several embedded event counters in IBM Power processors to derive correlations between different performance attributes: IPC, branch mispredictions, and cache misses. They also observed that these metrics exhibit periodicity, which can be exploited in the design of on-line table based history predictors. Using the periodicity of metrics they proposed asymmetric predictors that use one metric to predict another metric. The goal of their study was to design accurate table based predictors for programs with large variability. In particular, they did not focus on reducing simulation time and hence they did not present any methodology or automated tool to detect and exploit phase changes using embedded event counters.

3 The iPART Tool Description

To identify phases, we propose to correlate variations in program behavior with variations in execution frequencies of EIPs. One advantage of using EIPs is that the phase behavior identified by this approach is machine independent, since EIP execution frequencies are dependent only on the program inputs and compiler optimizations. In particular, execution frequencies are independent of the underlying machine configuration, such as the number of function units, and cache sizes.

This section describes iPART, a fully-automated tool that we developed to analyze phase behavior of programs running on native hardware. iPART is built on top of VTune, a commercially available software performance analyzer for Intel™ architectures. (more information on VTune is available online at `http://www.intel.com/software/products/vtune/`.)
iPART has the ability to non-intrusively analyze any application running on native hardware with negligible overhead, thereby alleviating the need for code instrumentation and-or recompilation required for several previously proposed methods.

A screen-shot of the iPART tool execution is shown in Figure 2. Some of the parameters of iPART are to select sampling rate, the size of instruction window for which an EIP vector is generated (see sec 4.1), and the target dimension for random projection (see sec. 4.2) to be used during clustering.

We modified the product version of VTune driver in order to collect the data and generate reports produced with iPART that are presented in this paper. For instance, our modifications remove aliasing of samples that simultaneously measure instructions retired and clockticks. We are considering making the complete iPART tool kit available to the public domain.

iPART uses a two step process to detect program phases. In the first step the underlying VTune driver monitors a large number of performance/code execution attributes stored in the embedded event counters of the Intel processors while a program is being executed on real hardware. It collects infor-

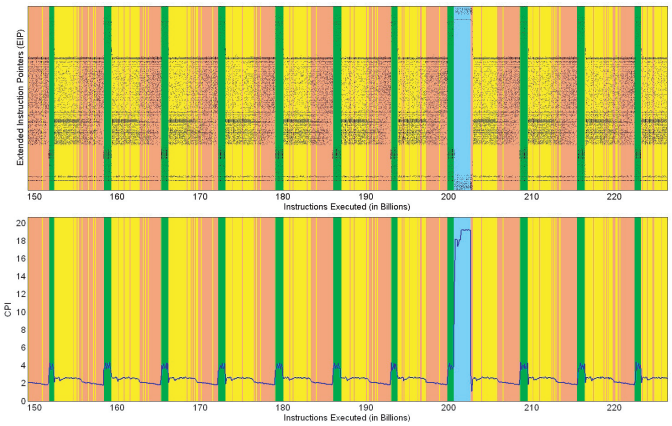

Fig. 5: **ammp:** Result of clustering of ammp with $k = 4$ clusters. Each cluster (or program phase) is color-coded. Observe from the lower figure that the temporal variations in program phases reflect well the temporal variations in performance (CPI.)

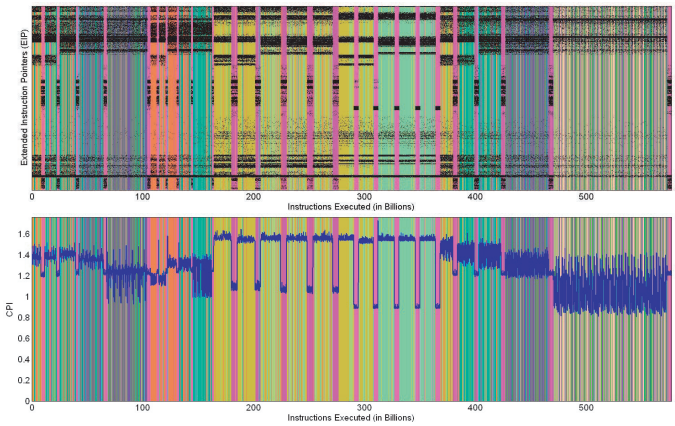

Fig. 6: **gzip:** Result of clustering of gzip with $k = 10$ clusters.

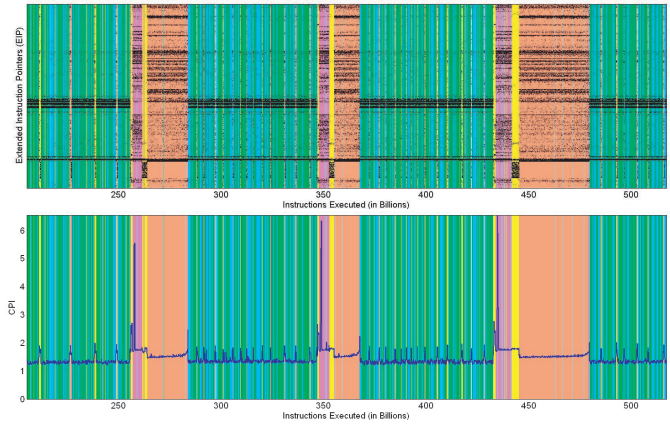

Fig. 7: **perlbmk:** Result of clustering of perlbmk with $k = 6$ clusters.

mation, such as EIPs and CPI, which are used in the second step to perform code clustering, phase analysis, and validation.

VTune interrupts execution at regular intervals of instructions executed (typically once every one million instructions) and records the EIP and event counter totals (e.g. clocktick count, instruction count.) Sampling at high frequency can significantly increase execution overhead. Conversely, too low a sampling frequency will lead to sparse data that could compromise phase analysis. Based on our experimental data, the default VTune sampling rate of once every one million instructions executed proved to be a good trade-off between execution overhead and collecting adequate sampling data; all results presented in this paper are sampled at that default rate. At this sampling rate, the typical overhead of using VTune is about 2%. For instance, ammp benchmark takes 516.5 seconds without VTune and 527.5 seconds with VTune, resulting in an acceptable 2.1% overhead.

Figure 3 shows the VTune-recorded EIPs and CPI of the full run of ammp (615 billion instructions.) This figure strongly suggests that even though VTune did not sample on every instruction, the default sampling rate is sufficient for correlating the CPI variations with EIP variations. Furthermore, this figure also shows that EIP is a valid attribute for detecting program phase behavior. Figure 4 shows a close up view of a segment of 80 billion instructions that shows a similar strong correlation between CPI and EIP even at a finer granularity.

4 Phase Detection Methodology

In this section, we present the methodolgy used by iPART in generating EIPVs from the samples collected by VTune. Then we present the k-means clustering algorithm which is similar to the one used by Sherwood *et al.* [14, 16], but applied to our novel EIP-based code vectors.

4.1 EIP Vectors and Similarity Metric

In order to construct EIPVs for clustering, the execution of a program is divided into equal intervals each of length 100 million instructions. Each interval is "signed" by a vector that corresponds to the normalized histogram of EIPs interrupted during its execution. Let N be the total number of unique EIPs recorded by VTune throughout the complete execution of the code. The j^{th} interval of 100 million executed instructions is then represented by the N-dimensional vector $\mathbf{x}_j = [x_{1j} \;\; x_{2j} \;\; \ldots \;\; x_{Nj}]^T$, where x_{ij} is the total number of times the i^{th} EIP has been sampled by VTune during the execution interval divided by the total number of EIPs collected (the vector $\mathbf{x}_j$ is normalized so that the sum of its entries x_{ij} equals one.) If VTune is set to sample code execution at its default rate of once every million instructions executed, then each histogram vector $\mathbf{x}_j$ is computed on the basis of 100 consecutive samples. We call $\mathbf{x}_j$ the j^{th} EIPV. Following this representation, the Euclidean norm in N-dimensional space is a natural distance metric to measure similarity between local code segments. In other words, the n^{th} and m^{th} EIPVs will be declared "similar" if the Euclidean distance $d(\mathbf{x}_n, \mathbf{x}_m) = \|\mathbf{x}_n - \mathbf{x}_m\|$ is "small". Note that since each EIPV covers a fixed number of instruction executed, it does not depend on performance metric (in particular, CPI) and EIPV is therefore a machine independent attribute.

4.2 K-means Algorithm and Random Projection

A given program run is represented by its set of N-dimensional EIP vectors $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$, where M is the total number of instruction segments executed (or total

number of instruction segments in units of 100 million.) The next step of iPART analysis consists of applying the k-means algorithm [7, 8] on the set $\mathcal{X}$ to identify the k most representative clusters corresponding to the k program phases. The k-means algorithm is an algorithm of linear complexity that is well suited for data clustering. However, it does not perform well for large dimensional spaces. More formally, k-means is known to work well only when the number of dimensions N is much smaller than the total number of vectors M. This problem is also known as the *dimensionality curse* of k-means. In the EIPV representation the dimensionality of the code space N is often comparable to, or larger than, the number of segments of 100 million instructions M. On average Spec2k benchmarks have 6524 vectors (see Table 1) and the number of unique EIPs is 4322. This suggests the need of reducing the dimensionality of the input vectors $\mathbf{x}_j$ before applying k-means clustering.

Random projection is a simple and effective way to reduce the dimensionality of the vectors without losing separability between clusters. Random projection consists of replacing the original set of EIPVs $\mathbf{x}_j$ ($j = 1, \ldots, M$) by their orthogonal projections $\mathbf{x}'_j$ onto a randomly selected linear subspace of dimension $D \ll N$. Intuitively, this corresponds to taking a random D-dimensional "slice" of the large N-dimensional code space. In [14] they also used k-means and random projection to cluster the basic block vectors obtained by code instrumentation. Similarly to their selection, we also select $D = 15$ as a suitable target dimension for random projection. K-means clustering is then applied to the set of "projected" EIPVs $\mathcal{X}' = \{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_M\}$.

Figure 5 shows, in a color-coded fashion, the result of clustering of ammp with $k = 4$ clusters (only the close-up view of 80 billion instructions is displayed.) The upper figure shows the clustering result in EIP space (the space used for clustering), while the lower figure shows the colored clusters superimposed with the instantaneous CPI curve. The strong correlation between clusters generated by EIPVs and temporal variations of CPI suggests that EIPV clustering is effective in identifying program phases for the purpose of deriving an accurate CPI.

Figures 6 and 7 show the same pictorial clustering results for two other representative Spec2k benchmarks gzip and perlbmk with $k = 10$ and $k = 6$ clusters, respectively. These figures provide additional evidence suggesting that an EIPV-based phase analysis is a promising approach to estimate the overall CPI.

It is worth mentioning that we also experimented extensively with two other clustering algorithms that are algorithmically more complex but do not require any dimensionality reduction: Spectral Clustering [11], and Agglomerative Clustering [9, 8]. Both algorithms yield almost identical clustering results to that of k-means with dimensionality reduction. Other than being useful information for validation, this experimental result further demonstrates that our choice of EIPVs produces well separated clusters that are easy to identify using a simple clustering algorithm such as k-means. Any further report on this comparison of clustering algorithms is beyond the scope of this paper.

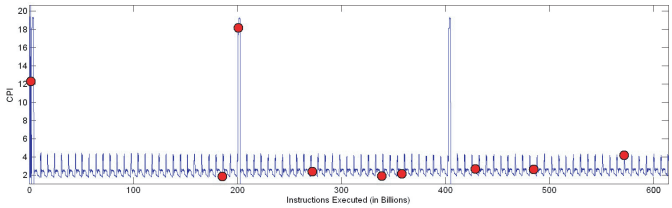

Fig. 8: **ammp:** The extracted simulation points for ammp with $k = k_{\text{BIC}} = 9$ clusters.

4.3 Bayesian Information Criterion

In the previous section we showed that the EIPV approach is effective in identifying phase behavior. In this section we use Bayesian Information Criterion (BIC) [13] to compute the number of distinct phases k. The Bayesian Information Criterion is an absolute measure of goodness of fit that accounts for model complexity by penalizing model configurations with large k (or equivalently large number of parameters to estimate.) Hence, we use the Bayesian Information Criterion (BIC) to automatically identify a good value for the number of clusters k. For a given choice of k the BIC score for an EIPV $\mathcal{X}' = \{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_M\}$ may be written as follows:

$$BIC(k) = \log\left(\widehat{p_k}(\mathcal{X}')\right) - \frac{k(D+1)}{2}\log M \qquad (1)$$

where $\widehat{p_k}(\mathcal{X}')$ is the probability of the data with estimated parameters, and $k(D+1)$ is the total number of parameters of our model (see [13].) Such a quantity is meant to take into account both accuracy (loglikelihood term) and complexity (penalty term) of the model. Adopting the exact same formulation as in [13], we model the probability distribution $\widehat{p_k}$ as a linear superposition of spherical Gaussians centered at every centroid $\mathbf{c}_1, \ldots, \mathbf{c}_k$ of the clusters (in D-dimensional space.) Subsequently, we define the good tradeoff number of clusters, k_{BIC}, as the value of k that maximizes $BIC(k)$. Column 5 of Table 1 reports the set of k_{BIC} for all the Spec2k benchmarks. Our results show that on average there are 9.2 phases in Spec2k benchmarks.

5 Performance estimation using Simulation Points

As stated earlier, the primary goal of this research is to exploit the phase behavior to reduce simulation time by eliminating simulation of repeated phases, without unduly impacting the simulation accuracy. After code clustering, each one of the k extracted program phases corresponds to a distinct code behavior. The next step consists of deriving from this analysis an accurate estimator for global performance (average CPI) that requires performance information over a minimal portion of the complete execution run.

We evaluate two different strategies for performance estimation. The first one consists of picking a single representative sample segment per phase. This strategy works well when the CPI is uniform within each program phase. While most benchmarks satisfy this condition, our results shows that a few benchmarks exhibit significant intra-phase CPI variance, leading to potentially large estimation errors using one-

				iPART Estimator (EIP-based clustering)				Uniform Sampling			
Name	Length	# of EIP	CPI true	k_{BIC}	CPI est.	error (in %)	σ_{EIP}	CPI est.	error (in %)	$\sigma(k)$	ratio
ammp	6146	3090	2.7005	9	2.6950	0.2065	0.0918	2.4112	10.7125	0.5836	6.3570
applu	28690	14235	0.8986	19	0.8783	2.2604	0.0354	0.8851	1.5024	0.0544	1.5363
apsi	18308	8106	1.4765	20	1.4758	0.0503	0.0423	2.2356	51.4099	0.1894	4.4800
art	2438	662	5.6073	18	5.6067	0.0105	0.0206	5.6030	0.0758	0.0400	1.9375
bzip2	5655	2492	2.3611	20	2.3666	0.2336	0.0795	2.5673	8.7315	0.1817	2.2869
cc1	2042	55305	2.1453	20	1.9354	9.7830	0.8526	2.2199	3.4769	0.5071	0.5947
crafty	2880	9897	1.9334	2	1.9248	0.4469	0.0472	1.9531	1.0188	0.0475	1.0058
eon	11126	6899	1.6450	2	1.6330	0.7313	0.0421	1.6522	0.4340	0.0426	1.0118
equake	2678	1261	2.2261	2	2.1268	4.4607	0.2022	2.1860	1.7995	0.2104	1.0405
facerec	14891	5354	1.6472	15	1.6341	0.7950	0.0814	1.4885	9.6317	0.1231	1.5138
fma3d	8732	8900	1.7182	11	1.6910	1.5811	0.1376	1.7422	1.3993	0.1258	0.9146
galgel	15247	4279	1.2419	19	1.2569	1.2120	0.0384	1.2495	0.6159	0.0836	2.1776
gap	2132	4611	1.5021	15	1.4616	2.6959	0.0600	1.5087	0.4436	0.1030	1.7182
gzip	5770	1910	1.3068	19	1.3225	1.1992	0.0223	1.3349	2.1498	0.0496	2.2221
lucas	6002	9935	1.4627	18	1.4657	0.2002	0.0349	1.4693	0.4509	0.0938	2.6869
mcf	1142	917	7.8716	20	7.9873	1.4701	0.1965	7.3404	6.7488	0.8840	4.4988
mesa	4771	2558	2.1776	7	2.1846	0.3216	0.0258	2.1518	1.1839	0.0354	1.3755
mgrid	65421	2900	0.6390	14	0.6399	0.1418	0.0075	0.6286	1.6328	0.0113	1.4972
parser	5075	7864	1.7938	2	1.7856	0.4553	0.2734	1.7094	4.7051	0.3727	1.3635
perlbmk	5849	12494	1.4679	10	1.4892	1.4517	0.0646	1.4520	1.0811	0.0925	1.4314
sixtrack	20596	3666	0.7640	6	0.7600	0.5270	0.0101	0.7531	1.4267	0.0240	2.3717
Swim	6570	695	2.5250	20	2.5298	0.1888	0.0732	3.1191	23.5305	0.3648	4.9864
twolf	4830	3382	3.7071	2	3.8030	2.5863	0.0778	3.6994	0.2088	0.0788	1.0138
vortex	4381	11071	1.3507	3	1.3125	2.8284	0.1614	1.2359	8.4998	0.1966	1.2185
vpr	3097	3210	3.0820	16	3.1124	0.9849	0.0800	3.1446	2.0320	0.0847	1.0597
wupwise	14061	3297	1.1678	9	1.1663	0.1281	0.0123	1.1711	0.2853	0.0217	1.7610
Geometric means	6524.3	4322.0	1.8363	9.2	1.8238	0.6154	0.0587	1.8591	1.8742	0.1022	1.7424

Table 1: **Spec2k Benchmark Results one-sample-per-phase iPART CPI estimate vs. uniform sampling:** The benchmark name is found in Column 1. The total length of each benchmark (in units of 100 million instructions) is reported in Column 2. Column 3 is the total number N of unique Extended Instruction Pointers (EIPs) sampled by VTune. The ground truth CPI is found in Column 4. Columns 5 to 8 report the results of approximating average CPI using the k simulation points (each 100M instruction long) computed by iPART. The number of clusters $k = k_{\text{BIC}}$ that maximizes the Bayesian Information Criterion (BIC) is reported in Column 5 and the resulting CPI estimation is shown in Column 6 (eq. 2.) The error of approximation (difference between Columns 4 and 6) is reported in Column 7. The confidence bound σ_{EIP} of the iPART estimator is in Column 8 (eq. 3.) For comparison, Columns 9 to 11 report the results of estimating the average CPI based on $k = k_{\text{BIC}}$ sample points picked uniformly in the code. Column 9 is the estimated CPI, Column 10 reports the error of estimation (difference between Columns 9 and 4), and Column 11 is the confidence bound of uniform sampling $\sigma(k)$. Lastly, Column 12 is the ratio of the confidence bounds of both the estimators: ratio $= \sigma(k)/\sigma_{\text{EIP}}$. Observe that for the majority of the benchmarks, the confidence ratio is larger than 1 indicating that code clustering in EIP space significantly reduces uncertainty in CPI estimation. The benchmarks for which the ratios are less than or equal to one do not exhibit any strong phase behavior (e.g. crafty, eon, twolf.) For those cases, both estimators return approximations that are equally reliable. The geometric means for all the columns are found in the last row of the table.

sample-per-phase strategy. In order to overcome this limitation, we also evaluate an alternative strategy, *stratified sampling*. In stratified sampling, a fixed total budget of samples is optimally allocated to each program phase according to the relative variability of performance within each phase in order to maximize confidence.

5.1 One sample per phase

The first approach (similar to [15]) is to select k segments (or simulation points) of 100 million instructions one per phase, and approximate the global average CPI as a weighted average of the CPI at the k simulation points. In order to maximize the likelihood that the each sample accurately represent the program phase that it represents, the sample is selected so that it is the closest to the centroid of its cluster in code space. This strategy is supported by the fundamental assumption that closeness in code space leads to closeness in performance. In other words, we assume that the performance is *locally constant* in code space.

Let $\{\mathbf{s}_1, \ldots, \mathbf{s}_k\}$ be the k simulation points and let M_p be the number of code segments (or points $\mathbf{x}_j$) belonging to cluster p (as a result, the sum of all M_p's equals M.) The p^{th} simulation point $\mathbf{s}_p$ is selected as the q^{th} instruction segment $\mathbf{x}_q$ where $\mathbf{x}_q$ is the point in cluster p that is the closest to the centroid $\mathbf{c}_p$. Subsequently, we derive the iPART estimator CPI_{EIP} for the global average CPI as the following weighted sum of the CPI at the simulation points $\mathbf{s}_p$:

$$CPI_{\text{EIP}} = \sum_{p=1}^{k} \alpha_p \, CPI(\mathbf{s}_p) \tag{2}$$

where the weight $\alpha_p = M_p/M$ is the relative population of cluster p.

Figure 8 shows the locations of the selected simulation points for the Spec2k benchmark ammp for $k = k_{\text{BIC}} = 9$ superimposed on the CPI space. Column 6 of Table 1 reports the results of CPI approximation for all the Spec2k benchmarks at k_{BIC}. The percentage errors between ground truth CPI (total number of clockticks measured by VTune divided by the total number of executed instructions) and estimated CPI are found in the column labeled "Error in %" of Table 1.

5.2 Measuring Robustness of CPI Estimation

While it is necessary to have a small CPI estimation error, we believe that it is not sufficient to show the robustness of EIPV approach. For instance, the error could be small when

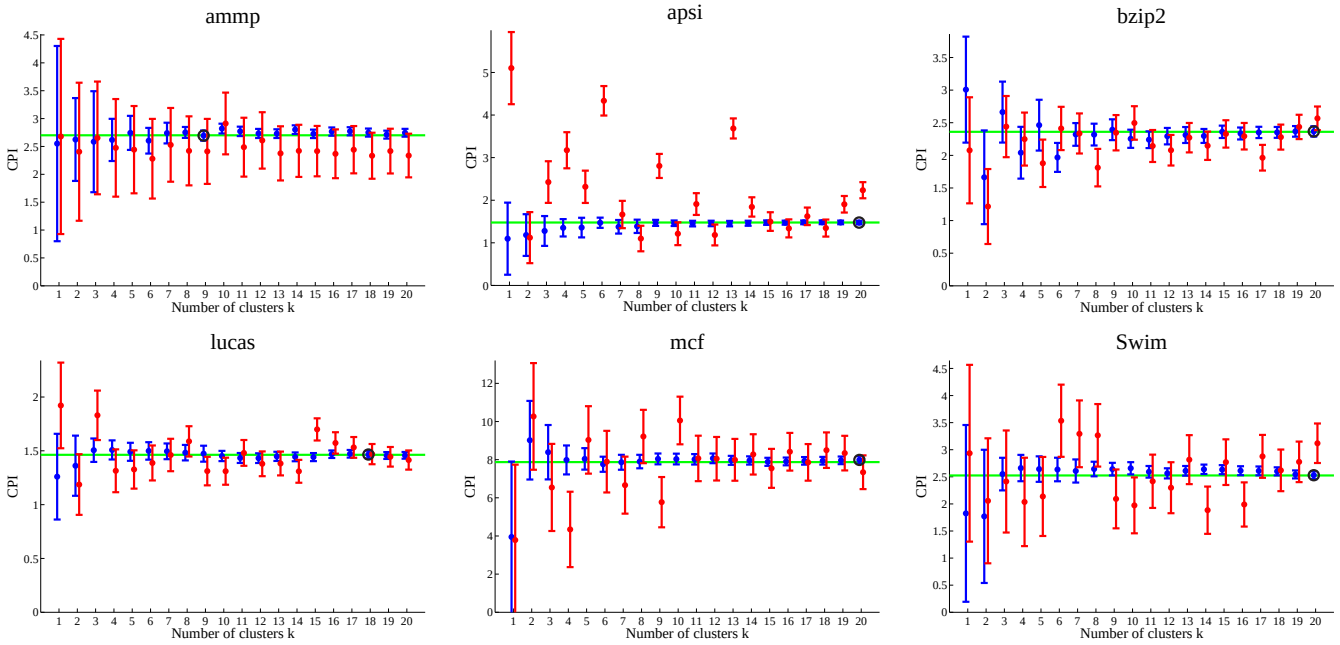

Fig. 9: **Estimation errors of the iPART and uniform sampling estimators at different values of k:** Representative cases that demonstrate superiority of iPART over a uniform sampling estimator. Estimation results of iPART are shown on the left (in blue) while the results of uniform sampling are on the right (in red.)

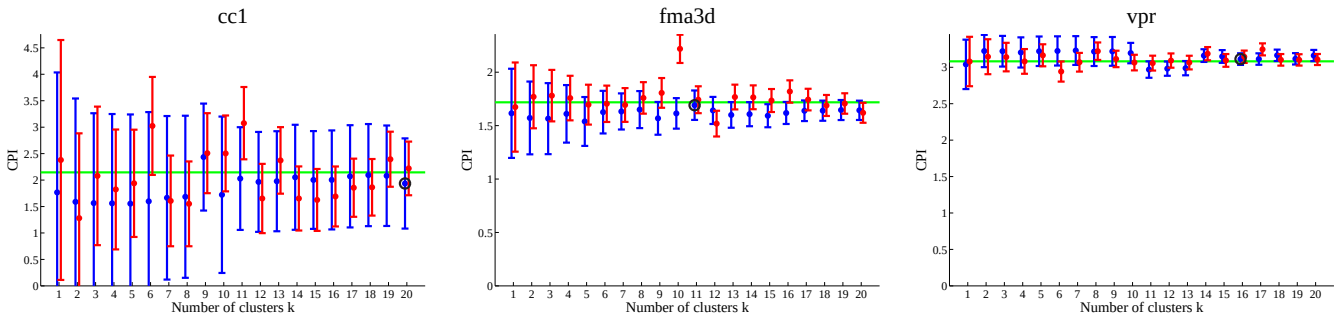

Fig. 10: **Estimation errors of the iPART and uniform sampling estimators at different values of k:** Several Spec2k benchmarks that exhibit non-conforming error profiles.

one"accidently" picks a single EIPV sample closest to the average cluster CPI even when the CPI varies significantly with in that cluster. Hence, CPI estimation error alone does not represent the true reliability level of the estimator. In order to address estimator reliability, we propose to compute the standard deviation σ_{EIP} of the iPART estimator. Evaluating the reliability of the iPART estimation method is essential for two reasons. The first reason is to measure robustness and repeatability, which corresponds to knowing if a given accuracy of estimation is likely to be reproduced if the same experiment were repeated several times under slight environmental perturbations. The second reason is to statistically compare the phase-based estimator against simpler estimators that do not require as much analysis effort.

Starting from Equation (2), we derive the following expression:

$$\sigma_{\text{EIP}}(k) = \sqrt{\sum_{p=1}^{k} \alpha_p^2 \, \sigma_p^2}, \qquad (3)$$

where σ_p^2 is the variance of the CPI values of all the samples in cluster p. Note that a small value of σ_p directly corresponds to a tight cluster in CPI space, which implies that every interval of execution that is part of that cluster exhibits similar program behavior. Conversely, if the code samples of each program phase do not map to tightly clustered CPI values, the corresponding cluster variance σ_p^2 may be large, and in the limit equal to the variance σ^2 of CPI values of the entire code population.

5.3 Comparison of iPART With Uniform Sampling

We now compare the variance of the EIPV approach used in iPART with a simple uniform sampling technique suggested by [17]. Uniform sampling estimates the performance by averaging the CPI values at k uniformly selected points in the code. The standard deviation $\sigma(k)$ of this estimator is given in Equation (4.) As expected, the confidence interval of the k-uniform sampling estimator decreases as the number of samples k increases. Following this confidence analysis, we can infer that the phase extraction approach provides benefits to performance estimation if and only if σ_{EIP} is smaller than $\sigma(k)$ for a given value of k, or alternatively, if the ratio $=$

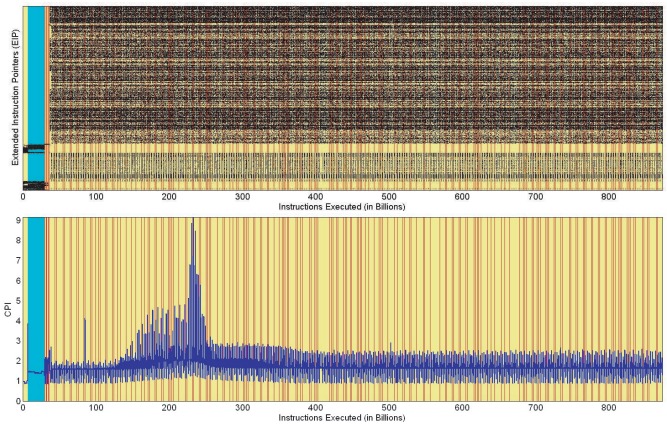

Fig. 11: **fma3d:** Result of clustering of fma3d with $k = 3$ clusters. Observe that the large yellow phase presents a lot of variability in CPI values.

$\sigma(k)/\sigma_{\text{EIP}}(k)$ exceeds one, where

$$\sigma(k) = \frac{\sigma}{\sqrt{k}}. \tag{4}$$

In order to demonstrate the benefits of phase extraction for the purpose of CPI estimation, we report a detailed comparison of performance of both iPART and k-uniform sampling estimators for all the Spec2k benchmarks. The experimental results of that comparison are found in Table 1 which shows the errors of approximation of the two estimators, the standard deviations $\sigma_{\text{EIP}}(k)$ and $\sigma(k)$ and the confidence ratios $= \sigma(k)/\sigma_{\text{EIP}}(k)$. From the last column in the table it is clear that for a vast majority of the benchmarks phase extraction provides clear benefits for CPI estimation than uniform sampling.

Figures 9 and 10 show the profiles of the iPART estimation CPI_{EIP} (on the left) and the uniform sampling estimator (on the right) as a function of the number of clusters k for a select few Spec2k benchmarks. In these figures, the horizontal line indicates the ground truth CPI, and the error bars correspond to the 65% confidence intervals centered at the estimate points, of total height $2\sigma_{\text{EIP}}(k)$ and $2\sigma(k)$ (Eq. (3),(4).) The black circles (larger dots) indicate the estimated CPI results reported in Table 1 at $k = k_{\text{BIC}}$.

Figure 9 shows six representative benchmarks for which phase analysis is clearly superior to uniform sampling. On these curves, the error bars of iPART are systematically smaller than that of the uniform sampling estimator, and decrease expectedly as k increases. This is because the variance of CPI in each cluster is considerably smaller than the overall CPI variance, hence yielding a high confidence ratio $\sigma(k)/\sigma_{\text{EIP}}(k)$ (between 2 and 6.) This shows that the locally constant hypothesis is clearly verified for those benchmarks. In addition, for all those cases, the ground truth CPI is always within the confidence intervals guaranteeing a small absolute error of estimation. Observe that the uniform sampling estimator on apsi exhibits a particularly erratic behavior across all values of k. This is explained by the fact that this benchmark is strongly periodic at high frequency, and therefore applying systematic sampling on the CPI curve introduces a large amount of aliasing leading to unreliable estimates. On the other hand, iPART does not fail since the various periodic code regions are explicitly separated in different program phases. The same periodic phenomenon is observed for mcf, Swim and bzip2 in Figure 9.

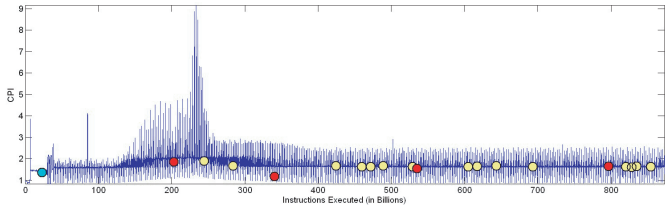

Fig. 12: **fma3d:** The budgeted $\nu = 20$ samples are allocated optimally within the $k = 3$ clusters. The color of the dots correspond to the color of the clusters in Figure 11. Observe that the yellow cluster exhibiting most of CPI variability is now allocated most of the samples (15), while the other two only have 1 and 4 samples.

Our results show that iPART on average provides 1.7 times higher confidence in the CPI estimation compared to uniform sampling. It also shows when each phase behaves uniformly in performance space a single sample is enough to accurately represent the entire phase behavior.

5.4 Stratified Sampling: Improving Confidence for Large Intra-Phase Variance

The benchmarks in figure 10 clearly denote insufficient performance of the one-sample-per-phase estimation. As the value of k increases the CPI variance does not decrease in these benchmarks. In fact, a closer look reveals that these benchmarks only exhibit a *partial* phase-based behavior. Look for example at fma3d (clustered at $k = 3$) in Figure 11: one of the phases towards the beginning of the code presents a very regular behavior, whereas the predominant phase has much more variability in CPI performance. This leads to a large estimator variance and therefore a low confidence ratio (see red dash curve in Figure. 14 for fma3d at $k = 3$.) In the section we propose an alternative estimator that overcomes this limitation, and maintains a high confidence ratio for all benchmarks.

Stratified sampling is an effective solution to reducing the variance of CPI estimation when the intra-cluster CPI variance is high. The ultimate objective is to optimally select a number of representative code segments that maximizes reliability of performance estimation while minimizing simulation time.

In practice, there is often a bound in the amount of time that could be spent in simulation, or alternatively in the total number of instructions that could be simulated. In this section we reformulate the estimation problem in terms of optimal resource allocation, namely, we answer the following question: *if we could simulate only up to N instructions, which ones should we pick given that we have performed phase detection?*

Because of the nature of our data, the question translates into optimally allocating a certain number of EIPV samples, each one representing 100 million instructions, across the different clusters.

Let us denote by ν the total number of samples allocated. In our experimental settings, we choose $\nu = 20$, which corresponds to an estimation of CPI based on a total of 2 billion instructions. The samples are distributed among the k clusters according to a partition $(\nu_1, ...\nu_k)$ of ν (i.e. $\sum_i \nu_i = \nu$.)

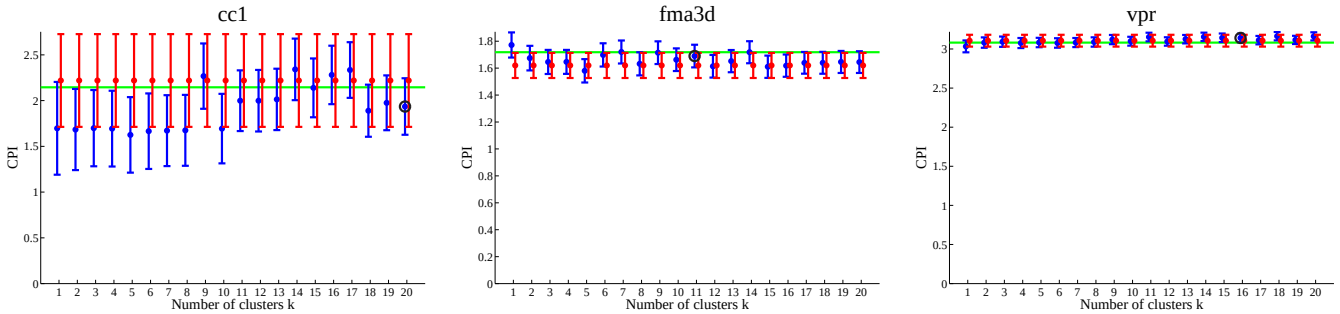

Fig. 13: **Estimation errors of stratified iPART (with budget of $\nu = 20$ samples) and uniform sampling estimators at different values of k:** Several Spec2k benchmarks whose error profiles clearly benefit from stratified sampling. Estimation results of stratified iPART are shown on the left (in blue) while the results of uniform sampling are on the right (in red.) Observe that every estimation point is now computed on the basis of the total budget of $\nu = 20$ points. This explains why the red error bars do not vary with the cluster number k. The ratios of sizes of the error bars are shown in Figure 14 and reported in the last column of Table 2.

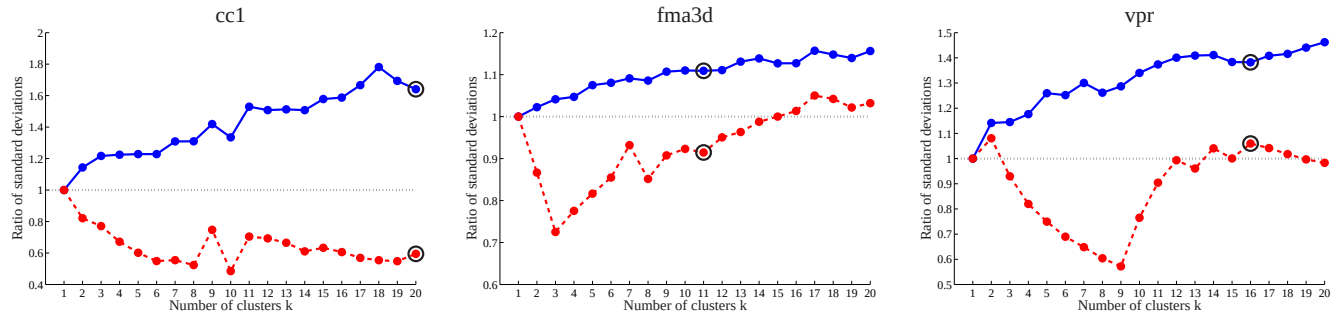

Fig. 14: **Estimation errors of the iPART and uniform sampling estimators at different values of k:** On varying k, we quantify the benefits of stratified sampling via the ratios of the standard deviations of the uniform estimator versus the phase-based estimators. The red curves show the ratios of the one-sample estimator. The blue curves show the ratios of the stratified estimator. Observe that stratified iPART always guarantees a confidence ratio larger than one, confirming a clear benefit of phase analysis over uniform sampling.

We apply a statistical technique called *stratified sampling* [2]. The CPI of each phase is first estimated as the average instantaneous CPI at the ν_i samples belonging to that phase. The representative samples $s_{i,1}, ...s_{i,\nu_i}$ are picked as the closest points to the centroid of the cluster in EIPV space.

The estimated CPI for phase i is therefore

$$CPI_{\text{EIP},i} = \frac{1}{\nu_i} \sum_{j=1}^{\nu_i} CPI(\mathbf{s}_{i,j}) \tag{5}$$

Then, the global CPI is estimated, similarly to Eq. 2, as the weighted average of the estimated CPI's of the different phases. The weights are still the phase population proportions.

$$CPI_{\text{EIP}} = \sum_{p=1}^{k} \alpha_p \, CPI_{\text{EIP},p} \tag{6}$$

We now address the issue of determining the optimal partition $(\nu_1, ...\nu_k)$ of ν. To this end, let us consider the standard deviation of our new stratified estimator (note the similarity with Eq. 3):

$$\sigma_{st\text{EIP}}(k) = \sqrt{\sum_{p=1}^{k} \alpha_p^2 \, \frac{\sigma_p^2}{\nu_p}}. \tag{7}$$

We are seeking the optimal allocation as the one minimizing the variance (also referred to as the optimal Neyman allocation [2].) A simple calculation shows that $\sigma^2_{st\text{EIP}}(k)$ is minimal when

$$\nu_i = \frac{\sigma_i \alpha_i}{\sum_{p=1}^{k} \sigma_p \alpha_p} \nu. \tag{8}$$

Observe that each ν_i depends not only on the cluster population proportion α_i, but also the standard deviation σ_i of the performance metric CPI within the i^{th} phase. In practice, each ν_i is approximated to its closest integer and always constrained to $\nu_i > 0$ so that each phase has at least one representative.

Substituting the optimal expression for ν_i in Eq. 7, the expression for the standard deviation simplifies into:

$$\sigma_{st\text{EIP}}(k) = \frac{\sum_{p=1}^{k} \alpha_p \sigma_p}{\sqrt{\nu}}. \tag{9}$$

As a result, the confidence ratio for stratified sampling becomes:

$$\text{ratio}_{st\text{EIP}}(k) = \frac{\sigma(\nu)}{\sigma_{st\text{EIP}}(k)} = \frac{\sigma}{\sum_{p=1}^{k} \alpha_p \sigma_p}. \tag{10}$$

Observe that the stratified confidence ratio does not depend upon the budget of sample points ν, but only the class proportions α_p, the class standard deviations σ_p and standard deviation of the whole population σ. As a result, it can be interpreted as a direct quantitative assessment of the benefit provided by phase analysis for the purpose of performance estimation.

Figure 12 shows how stratified sampling acts on fma3d at $k = 3$: the phase that exhibits more variability in CPI is now represented by several samples (the phase depicted in yellow in Figure 11 has 15 samples out of 20.) The percentage error decreases, and the confidence in our results increases. The error bars for fma3d and a few other representative benchmarks

		iPART Stratified Estimator (EIP-based clustering)				Uniform Sampling			
Name	CPI true	k_{BIC}	CPI est.	error (in %)	σ_{EIP}	CPI est.	error (in %)	$\sigma(\nu)$	ratio
ammp	2.7005	9	2.6764	0.8933	0.0584	2.3368	13.4686	0.3915	6.7036
applu	0.8986	19	0.8783	2.2604	0.0249	0.8845	1.5695	0.0530	2.1319
apsi	1.4765	20	1.4758	0.0503	0.0325	2.2356	51.4099	0.1894	5.8282
art	5.6073	18	5.6067	0.0105	0.0185	5.5409	1.1839	0.0379	2.0519
bzip2	2.3611	20	2.3666	0.2336	0.0616	2.5673	8.7315	0.1817	2.9479
cc1	2.1453	20	1.9354	9.7830	0.3091	2.2199	3.4769	0.5071	1.6406
crafty	1.9334	2	1.9309	0.1325	0.0149	1.9423	0.4592	0.0150	1.0062
eon	1.6450	2	1.6402	0.2949	0.0133	1.6469	0.1144	0.0135	1.0120
equake	2.2261	2	2.2055	0.9211	0.0604	2.1690	2.5634	0.0665	1.1020
facerec	1.6472	15	1.6454	0.1062	0.0548	1.6753	1.7069	0.1066	1.9467
fma3d	1.7182	11	1.6897	1.6577	0.0842	1.6199	5.7177	0.0933	1.1091
galgel	1.2419	19	1.2569	1.2120	0.0189	1.1143	10.2690	0.0815	4.3111
gap	1.5021	15	1.4782	1.5916	0.0433	1.4441	3.8583	0.0892	2.0617
gzip	1.3068	19	1.3225	1.1992	0.0197	1.3251	1.4045	0.0483	2.4497
lucas	1.4627	18	1.4657	0.2002	0.0259	1.4142	3.3211	0.0890	3.4366
mcf	7.8716	20	7.9873	1.4701	0.1490	7.3404	6.7488	0.8840	5.9324
mesa	2.1776	7	2.1652	0.5684	0.0135	2.1929	0.7056	0.0210	1.5479
mgrid	0.6390	14	0.6397	0.1046	0.0060	0.6389	0.0231	0.0095	1.5829
parser	1.7938	2	1.7586	1.9614	0.0857	1.6521	7.9003	0.1179	1.3745
perlbmk	1.4679	10	1.4542	0.9316	0.0361	1.4907	1.5547	0.0654	1.8116
sixtrack	0.7640	6	0.7623	0.2200	0.0046	0.7535	1.3690	0.0131	2.8543
Swim	2.5250	20	2.5298	0.1888	0.0646	3.1191	23.5305	0.3648	5.6473
twolf	3.7071	2	3.7452	1.0274	0.0240	3.7158	0.2336	0.0249	1.0381
vortex	1.3507	3	1.3744	1.7494	0.0562	1.2867	4.7387	0.0761	1.3547
vpr	3.0820	16	3.1415	1.9301	0.0548	3.1065	0.7929	0.0758	1.3823
wupwise	1.1678	9	1.1732	0.4635	0.0069	1.1692	0.1159	0.0145	2.1196
Geometric means	1.8363	9.2	1.8277	0.5324	0.0326	1.8478	1.9643	0.0695	2.1335

Table 2: **Spec2k Benchmark Results stratified iPART CPI estimate (with budget of $\nu = 20$ samples) vs. uniform sampling:** Similarly to Table 1, Columns 3 to 6 report the CPI estimation results of stratified iPART. Column 4 show the stratified estimation of CPI following Equation 6 and the standard deviations of the stratified iPART estimator is reported in Column 6 (eq. 9.) Observe that the percent errors reported in Column 5 are either similar or slightly smaller to that reported in Table 1. More importantly, the confidence ratios ratio$_{st\text{EIP}}(k)$ (Eq. 10) listed in the last column are significantly larger than those of the non-stratified estimator. This is the fundamental benefit of the stratified sampling strategy.

with large intra-cluster variance are shown in Figure 13. The advantage is clear when we compare this figure with the corresponding one-sample per phase Figure 10.

In Table 2 we report the results of the stratified CPI estimator for all the Spec2k benchmarks at k_{BIC}. Compare with the results of Table 1, we note that the confidence ratio$_{st\text{EIP}}(k)$ in the estimation is considerably higher. Observe that the benchmarks for which the one-sample-per-cluster strategy worked well (such as the ones shown in Figure 9) are still very well estimated under stratified sampling, with slightly higher confidence ratios. For example, the confidence ratio for apsi goes from 4.48 to 5.82 by applying stratified sampling.

The confidence ratios ratio$_{st\text{EIP}}(k)$ in the last column are now always larger than one. As shown in Figure 14, even for the challenging benchmarks, this happens for every k. This is not the case for the one-sample estimator. This means that the stratified sampling always guarantees a more reliable estimation of performance than uniform sampling (at comparable budget of simulation samples) while taking optimal advantage of the behavior phases computed through phase detection.

6 Practical Considerations for Exploiting Phase Behavior

While the primary goal of iPART is to detect program phases, the end goal of this research is to generate traces that capture the entire program behavior. To this end, the results from iPART can be used to generate multiple trace segments (one or more per phase), where each trace segment coincides with the EIPV representative chosen by iPART. However, phase detection and trace collection are not done at the same time and the two runs may not be identical when running on the native hardware. Hence, it is important for identified phases to be less sensitive to small run-time variations. One approach to achieve this goal is to minimize the number of phase changes for a given *k*.

The rigid approach of always building vectors with 100 million instructions will inevitably produce ambiguous vectors at phase boundaries that will have EIPs from adjoining phases. The clustering algorithm either treats these vectors to be part of a different phase, thereby increasing the k value, or may place these vectors into one of the existing phases, thereby increasing the error in CPI estimation.

To address the problems posed by ambiguous vectors at phase boundaries, in this section we present *QPhase*, which is a novel approach to split 100 million EIPVs at phase boundaries into smaller vectors that will have no more than 100 million instructions each.

QPhase identifies phase changes by tracking the locus of execution - the average memory location in the application's instruction space over a given period. For every EIP sample (S) collected by VTune, QPhase computes the arithmetic average of the immediately preceding 100 EIP samples to compute the *locus of execution*, and a similar locus is computed for the immediately following 100 EIP samples.

Finally, QPhase computes the absolute value of the differ-

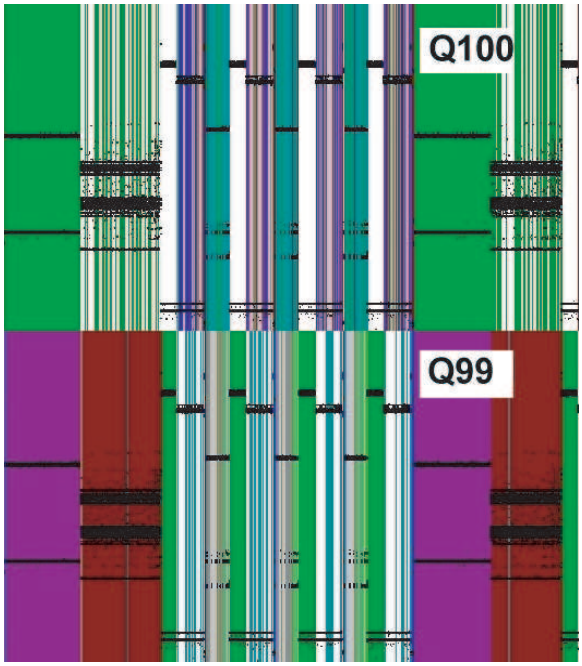

Fig. 15: **Galgel:** In the bottom half, QPhase with Q99 correctly isolates the first and second phases on the left and again on the right while the top half with Q100 builds an ambiguous vector that combines with 2 later phases. If we define the average phase width as the average number of consecutive vectors belonging to the same phase, then it is clear that the average phase width with Q99 is greater to that with Q100. This is characteristic advantage of Qphase: it produces more stable and solid phases.

ence between two loci (the derivative) and associates it with the current EIP sample S. The derivatives at each EIP sample are sorted in descending order and the top 1% (or quantile) are marked as possible phase change boundaries. We call this Q99 QPhase detection. In our initial experiments we chose 1% as the cut-off point and we plan to investigate the optimal value in the future. Note that specifying 0% quantile (Q100) turns off the QPhase algorithm. Since VTune samples every one million instructions, QPhase detects an abrupt change in locus of execution within a window of execution of 200 million instructions.

Phase change information can then be used to segregate the EIPs of one phase from the next by building a shortened vector that does not include any EIPs from the next phase. QPhase, however, does not prohibit the shortened vector from clustering with either the previous or the next phase. The shorter EIPVs are merely suggestions which a clustering algorithm can exploit for cleaner separation of the phases.

6.1 Illustration of QPhase Effectiveness

A fragment of a QPhase analysis is shown in Figure 15. The bottom half is using QPhase (Q99) while the top half is not (Q100.) Both figures are produced using $k = 10$ on the same data and using the same clustering algorithm. Both produced an estimate of CPI with about the same error. Note that QPhase effectively isolates the second phase from the left separating it completely from the previous and next phase each time that phase appears during execution. However, Q100 figure shows that the second phase is unnecessarily split into multiple phases indicating the inability of the clustering algorithm to separate out ambiguous EIPVs. Over the entire

		No Qhase	QPhase	Phase Width
Name	k	Phase changes (in %)	Phase changes (in %)	Increase
ammp	6	50.6	39.9	0.5292
applu	20	70.9	71.9	-0.0196
apsi	20	64.8	61.4	0.0847
art	5	71.3	20.3	3.5264
bzip2	11	65.6	62.4	0.0775
cc1	16	41.8	49.3	-0.3634
crafty	3	60.3	12.9	6.1178
eon	6	82.0	24.8	2.8099
equake	4	71.2	16.5	4.6548
facerec	15	78.4	78.1	0.0038
fma3d	6	61.1	24.6	2.4321
galgel	16	34.2	35.4	-0.0981
gap	14	70.1	65.7	0.0946
gzip	14	54.0	50.6	0.1249
lucas	12	60.8	62.3	-0.0402
mcf	13	55.1	50.6	0.1597
mesa	7	84.3	20.3	3.7488
mgrid	7	61.0	60.8	0.0058
parser	11	82.4	72.8	0.1604
perlbmk	15	69.8	66.9	0.0629
sixtrack	5	79.2	17.9	4.317
swim	11	82.1	69.9	0.2132
twolf	3	66.8	13.9	5.6899
vortex	4	47.6	28.4	1.4222
vpr	5	39.4	15.6	3.8822
wupwise	10	72.6	69.8	0.0549

Table 3: Spec2k In the majority of SPEC2000 benchmarks, QPhase has fewer phase transitions. The Phase changes % is the number of phase changes divided by the total number of vectors - the theoretical maximum possible number of phase changes. The Phase Width increase is in term of vectors. For instance, twolf has 5.7 more vectors per phase or about 500 million more instructions per phase. Notice that the k values appearing in this table were selected arbitrarily and are not necessarily equal to k_{BIC}.

Spec2k benchmark suite, Q99 generates 154k phase changes, while Q100 produces 176k phase changes.

In addition, it should be expected that the average phase width should be greater with Q99 if it is getting better separation. Table 3 shows this to be the case in most of the Spec2k benchmarks. In addition, Table 3 shows that in most cases QPhase produces fewer phase transitions. The benefit from QPhase is greater when k is small - see crafty or twolf. However, when k is high, QPhase makes less of a difference since EIPVs at the phase boundaries are more likely to be placed in a separate cluster at large k.

7 Conclusions and Future Work

The first step in phase analysis research is to develop a non-intrusive, and low overhead methodology for detecting program phase behavior. The encapsulation of this research is *iPART*, a fully-automated tool that is built on top of Intel's VTune performance analyzer. iPART creates a machine independent, time-sequenced set of code vectors composed of Extended Instruction Pointers. We demonstrate that this representation enables effective and stable program phase detection applying the well known k-means clustering algorithm. Moreover, we show that this approach is suitable for the purpose of deriving reliable and efficient CPI estimators that are on average 2 times more accurate than other natural estimators, such as random or systematic sampling. In this paper we present two different estimators derived from phase anal-

ysis. The first one consists of selecting a single sample per phase and approximate the global CPI as a weighted average of the CPI at the selected samples. The second method, stratified sampling, optimally allocates a fixed number of samples among different clusters, and averages their CPIs according to clusters sizes and variances, in order to minimize the overall variance of the estimator.

Using variance analysis this paper shows that for benchmarks with a large intra-cluster CPI variance, stratified sampling is a good alternative to the one-sample-per-phase approach to maximize confidence in performance estimation. While keeping estimation errors extremely low, stratified sampling increases the confidence ratio with respect to the uniform sampling estimator even in challenging benchmarks such as cc1 or fma3d.

We also show that using fixed width code vectors of 100 million instructions perturbs clustering dramatically when vectors span multiple phases. This paper presents QPhase, a novel approach that identifies phase changes by tracking the locus of execution. QPhase generates variable width vectors at phase boundaries to improve clustering results.

We believe that the simplicity, configurability, and ease of use of iPART will enable us to pursue several promising research directions in the future. Here are some specific areas that we are either currently investigating or plan to investigate:

- All our current investigations have only considered Spec2k benchmarks. For server class processor design it is necessary to increase the scope of our benchmarks. We are currently analyzing the phase behavior of commercial workloads (TPC-C, SpecJAppServer and TPC-H.) We also plan to investigate other emerging applications such as computer vision and machine learning.

- We plan to extend iPART infrastructure to other processor architectures, such as Itanium™. These extensions will allow us to understand how phase behavior changes across different platforms and architectures. Furthermore, these experiments can help us understand the machine independence of the code vectors across an entire family of processors.

- Can the operating system and platform behavior be correctly modeled with this phase analysis tool? Comparing the behavior of Linux and Windows may yield insight into the reality of program behavior outside of the limited simulation environment.

- The size of local code segments may shrink or expand beyond 100 million or may be variable. Is there an optimal value? Does that optimum depend on the benchmark under scrutiny?

- Other embedded processor counters may be useful for generating the EIPs for phase analysis, specifically the Branches Retired event. In addition to interrupting on the Branches Retired, VTune also has the ability to dump the branch table at the time of interrupt.

References

[1] R. Balasubramonian, D.H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 245–247, December 2000.

[2] P.T. Bickel and K.A. Doksum. Mathematical Statistics. Basic Ideas and Selected Topics. Vol I. Second Edition. Prentice Hall, 2001.

[3] A. Dhodapkar and J.E. Smith. Managing Multi-Configuration Hardware via Dynamic Working Set Analysis. In *Proceedings of the 29th International Symposium on Computer Architecture*, pages 233–244, May 2002.

[4] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and Predicting Program Behavior and its Variability. In *Proceedings of the 2003 International Conference on Parallel Architectures and Compilation Techniques*, September 2003.

[5] L. Eeckhout, D. Stroobandt, and K.D. Bosschere. Efficient Microprocessor Design Space Exploration through Statistical Simulation. In *Proceedings of the 36th Annual Simulation Symposium*, pages 233–240, March 2003.

[6] L. Eeckhout, H. Vandierendonck, and K.D. Bosschere. Workload Design: Selecting Representative Program-Input Pairs. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, September 2002.

[7] J.A. Hartigan and M.A. Wong. Algorithm AS136: A k-means Clustering Algorithm. In *Applied Statistics*, volume 28, pages 100–108, 1979.

[8] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2001.

[9] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 1990.

[10] A.J. KleinOsowski and D.J. Lilja. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *Computer Architecture Letters*, 1, June 2002.

[11] A. Y. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. In *Proceedings of NIPS 14*, 2002.

[12] S. Nussbaum and J.E. Smith. Modeling Superscalar Processors via Statistical Simulation. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 15–24, September 2001.

[13] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of number of clusters. In *Proceedings of the 17th International Conference on Machine Learning*, pages 727–734, 2000.

[14] T. Sherwood, E. Perelman, and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, September 2001.

[15] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, December 2002.

[16] T. Sherwood, S. Sair, and B. Calder. Phase Tracking and Prediction. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 336–347, June 2003.

[17] R.E. Wunderlich, T.F. Wenisch, B. Falsafi, and J.C. Hoe. SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 84–95, June 2003.